5/2/2019

Implementation of SNMP Protocol

For GetRequest PDU

Saurabh P Bhandari

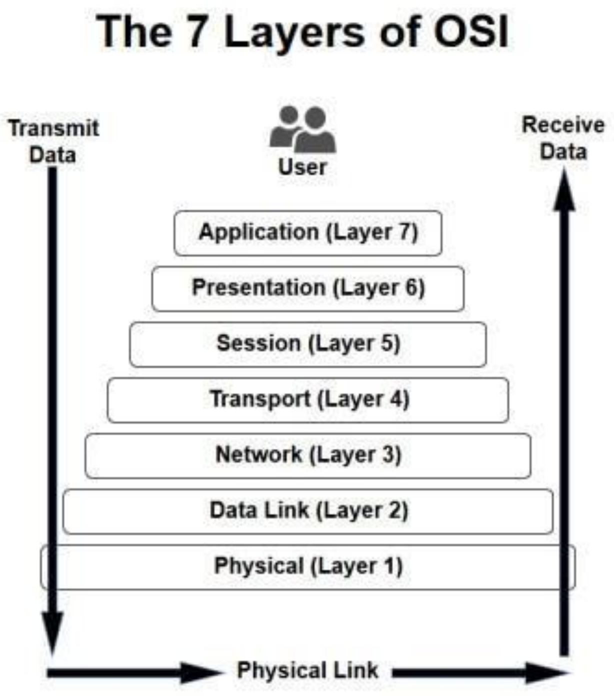
Introduction

OSI Model

The Open Systems Interconnection (OSI) model is a reference tool for understanding data communications between any two-networked systems. It divides the communications processes into seven layers. Each layer both performs specific functions to support the layers above it and offers services to the layers below it. The three lowest layers focus on passing traffic through the network to an end system. The top four layers come into play in the end system to complete the process.

The OSI model is a product of the Opens Systems Interconnection project at the International Organization for Standardization (ISO). The ISO-OSI model is a seven-layer architecture, which allows all network elements to operate together irrespective of the creator of the protocols and the computer vendors supporting the protocols.

In the OSI model, the application layer provides an interface for the end user operating a device connected to a network, This layer is what the user sees, in terms of loading an application (such as Web browser or e-mail); that is, this application layer is the data the user views while using these applications.



Network Management

Network management consists of all the administrative actions taken to keep a network running efficiently. The ISO and the International Telecommunications Union defines a formal model for telecommunications and network management. The original model defines five areas of concern (known as **FCAPS** after the first letter of each area):

- Fault
- Configuration
- Accounting
- Performance
- Security

Fault:

- **Device Management**: Monitoring of all switches, routers, servers and other network hardware to make sure they are running properly.
- **Server Management**: Monitoring of the network's application layer, that is, all network-based software services; these include login authentication, email, web servers, business applications and file servers.
- **Link Management**: Monitoring of long-haul links to ensure they are working.

Configuration:

- **Network Architecture**: The overall design, including topology, switching vs routing and subnet layout.
- **Configuration Management**: Arranging for the consistent configuration of large numbers of network devices.
- **Change Management**: How a site rolls out new changes, from new IP addresses to software updates and patches?

Accounting:

- Involves collecting network user data such as link utilization, disk drive or data storage usage, and CPU processing time.
- **Billing Management**: Using the statistics, the users can be billed and usage quotas can be enforced. These can be disk usage, link utilization, CPU time, etc.

Performance:

- **Traffic management**: Using the techniques of Queuing and Scheduling to allocate bandwidth shares among varying internal or external clients or customers.
- **Service-level management**: Making sure that agreed-upon service targets – *e.g.* bandwidth – are met (depending on the focus, this could also be placed in the fault category).

Security:

- Process of controlling access to assets in the network.
- Functions include managing network authentication, authorization, and auditing, such that both internal and external users only have access to appropriate network resources.

Simple Network Management Protocol (SNMP)

Simple Network Management Protocol (SNMP) is an application-layer protocol used to manage and monitor network devices and their functions. SNMP provides a common language for network devices to relay management information within single and multivendor environments in a local area network (LAN) or wide area network (WAN).

Devices that typically support SNMP include cable modems, routers, switches, servers, workstations, printers, and more.

SNMP versus Management

While SNMP is a very important *tool* for network management, it is just a tool. Network management is the process of making decisions to achieve the goals outlined above, subject to resource constraints. SNMP simply provides some input for those decisions.

SNMP protocol allows a device to report information about its current operational state; for example, a switch or router may report the configuration of each interface and the total numbers of bytes and packets sent via each interface.

NETCONF/YANG

NETCONF/YANG, introduced by IETF (Internet Engineering Task Force) to overcome issues such as unreliability, insecurity, etc., faced in configuring of networking devices via CLI or SNMP, it provides a standardized way to programmatically update and modify the configuration of a network device.

YANG is the *modelling language* that describes the configuration changes.

Whereas, **NETCONF** is the *protocol* that applies the changes to the relevant data store (i.e. running, saved etc.) upon the device.

NETCONF

NETCONF (NETwork CONFiguration) is a protocol defined by the IETF to “install, manipulate, and delete the configuration of network devices”. NETCONF operations are performed via a RPC layer using XML based encoding.

Some of the key features to NETCONF are, ability to rollback configurations, ability to support any data model and the separation of config from operational state.

YANG

YANG (Yet Another Next Generation) is a data modelling language, providing a standardized way to model the operational and configuration data of a network device. YANG, being a language is being protocol independent, can then be converted into any encoding format, e.g. XML or JSON.

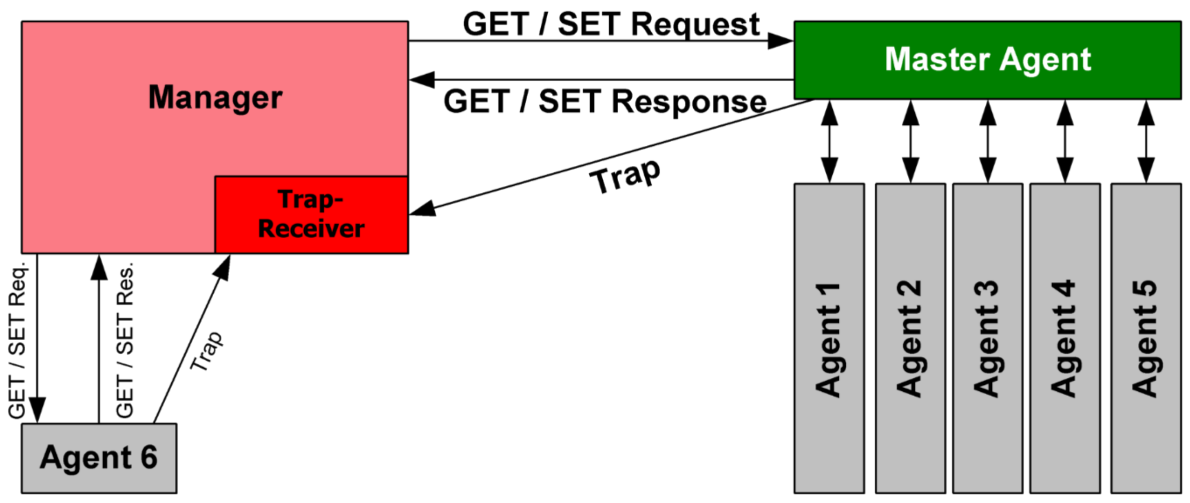
Introduction to SNMP

Overview

SNMP was introduced in 1988 and now includes three distinct versions SNMPv1, SNMPv2, and SNMPv3. SNMP is the protocol that allows an SNMP manager (the controller) to control an SNMP agent (the controlee) by exchanging SNMP messages. An SNMP message is a packet sent over UDP/IP to port 161. UDP/IP is the User Datagram Protocol over IP. An SNMP node that replies to requests for information is known as an **SNMP agent**. The network node doing the SNMP querying is known as the **manager**.

The main purpose of an SNMP message is to control (set) or monitor (get) parameters on an SNMP agent. In SNMP, a parameter is an instance of a more generic object. For example, an SNMP agent may have several instances of a microphoneMute object -- one instance for each microphone input. An SNMP manager can set or get the value for each instance (each parameter). In an SNMP agent, parameters are arranged in a tree. SNMP uses an Object Identifier (OID) to specify the exact parameter to set or get in the tree. An OID is a list of numbers separated by periods. For example, the OID addressing the microphoneMute parameter in a Rane NM 1 is '1.3.6.1.4.1.2680.1.2.7.3.2.0'. This OID is actually a combination of two values. The first value is the OID of the generic object '1.3.6.1.4.1.2680.1.2.7.3.2'. The second is the instance value, which specifies the particular instance of the mirophoneMute object.

Every SNMP agent has an address book of all its objects, called the MIB or Management Information Base. The MIB provides the name, OID, data type, read/write permissions, and a brief description for each object in an SNMP agent.



SNMP versions

SNMP has three official versions, SNMPv1, SNMPv2 and SNMPv3. SNMPv1 made its first appearance in 1988 in a collection of RFCs starting with **RFC 1065** (updated in **RFC 1155)**.

SNMPv2 was introduced in 1993 with **RFC 1441**. SNMPv2 expanded on the basic information, starting with **RFC 1442** (currently **RFC 2578**), and also introduced improved techniques for managing tables.

SNMPv2 also included a proposed security mechanism, but it was largely rejected by the marketplace. Ultimately, a version of SNMPv2 known as SNMPv2c that used the SNMPv1 “community” security mechanism was introduced.

SNMPv3 then finally delivered a model for reasonably effective security. The “User-based Security Model” or **USM** was first proposed in 1998 in **RFC 2264**.

ASN.1

All SNMP devices must understand an SNMP message, which presents a couple problems. The first problem exists because different software languages have slightly different sets of data types (integers, strings, bytes, characters, etc.). For example, an SNMP manager sending a message full of Java data types may not be understood by an SNMP agent written in C. To solve this problem SNMP uses ASN.1 or Abstract Syntax Notation One to define the data types used to build an SNMP message. Since ASN.1 is independent of any particular programming language, the SNMP agents and managers may be written in any language.

Constructing a message requires some knowledge of the data types specified by ASN.1, which fall into two categories: primitive and complex. ASN.1 primitive data types include Integer, Octet (byte, character) String, Null, Boolean and Object Identifier. To expand the programmer's ability to organize data, ASN.1 allows primitive data types to be grouped together into complex data types. Complex data types include Sequence (a list of data fields) and SNMP PDU (Protocol Data Unit) which contains the body of an SNMP message. Below are some of the ASN.1 data types:

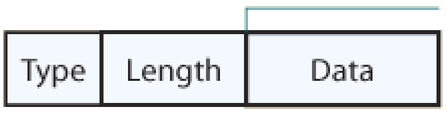
Primitive Data Types	**Identifier**	**Complex Data Types**	**Identifier**
Integer	0x02	Sequence	0x30
Octet String	0x04	GetRequest PDU	0xA0
Null	0x05	GetResponse PDU	0xA2
Object Identifier	0x06	SetRequest PDU	0xA3

Basic Encoding Rules

When using valid ASN.1 data types another problem still exists. When sending a particular data type over the wire, how should it be encoded? Should strings be null terminated as in the programming language C, or not? Should Boolean values be 8 bits as in C++ or 16 bits as in Visual Basic 6?

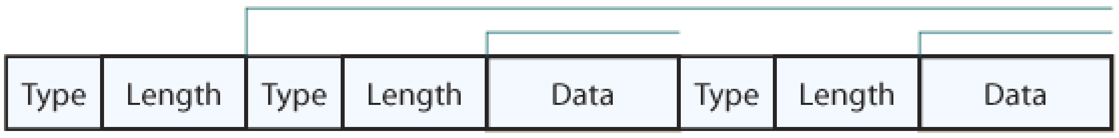
ASN.1 includes Basic Encoding Rules (BER) to address this problem. Regardless of programming language, all data types are encoded the same way before they are placed on the wire by following the Basic Encoding Rules. In short, all data fields in an SNMP message must be a valid ASN.1 data type, and encoded according to the BER.

The most fundamental rule states that each field is encoded in three parts: Type, Length, and Data. Type specifies the data type of the field using a single byte identifier. Length specifies the length in bytes of the following Data section, and Data is the actual value communicated (the number, string, OID, etc.).



Format: BER Encoded Field (Primitive Data Type)

Some data types, like Sequences and PDUs, are built from several smaller fields. Therefore, a complex data type is encoded as nested fields.

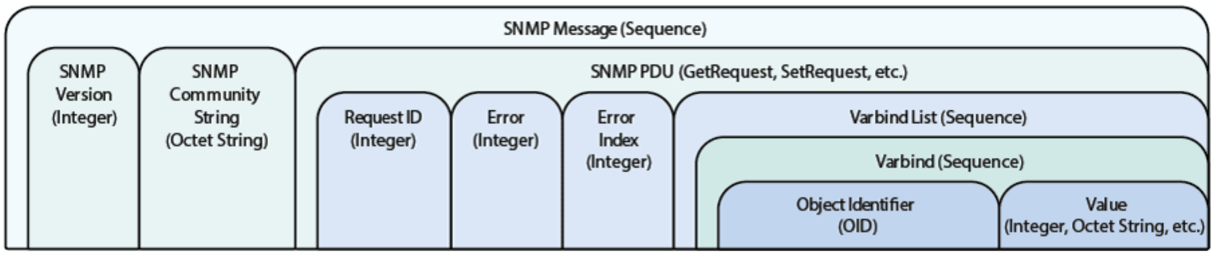


Format: BER Encoded Fields (Complex Data Type)

There are two more Basic Encoding Rules necessary for encoding an SNMP message. Both apply to encoding OIDs. The first rule applies when encoding the first two numbers in the OID. According to BER, the first two numbers of any OID (x.y) are encoded as one value using the formula (40*x)+y. The first two numbers in an SNMP OID are always 1.3. Therefore, the first two numbers of an SNMP OID are encoded as 43 or 0x2B, because (40*1)+3 = 43. After the first two numbers are encoded, the subsequent numbers in the OID are each encoded as a byte. However, a special rule is required for large numbers because one byte (eight bits) can only represent a number from 0-255. For example, the number 2680 in the Rane NM 1 microphoneMute OID '1.3.6.1.4.1.2680.1.2.7.3.2.0' cannot be encoded using a single byte. The rule for large numbers states that only the lower 7 bits in the byte are used for holding the value (0-127). The highest order bit is used as a flag to let the recipient know that this number spans more than one byte. Therefore, any number over 127 must be encoded using more than one byte. According to this rule, the number 2680 must be encoded 0x94 0x78. Since the most significant bit is set in the first byte (0x94), the recipient knows to use the lower 7 bits from each byte (0x14 and 0x78) and decode the two bytes as (0x14 *128) + 0x78 = 2680.

SNMP Message Format

The SNMP message format specifies which fields to include in the message and in what order.

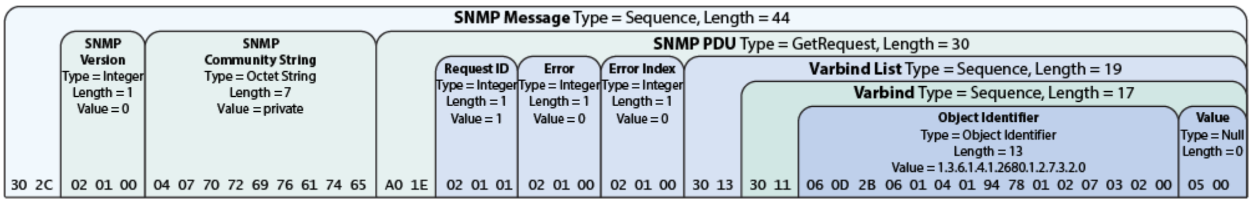


Field	Description
SNMP message	A Sequence representing the entire SNMP message consisting of the SNMP version, Community String, and SNMP PDU.
SNMP Version	An Integer that identifies the version of SNMP. SNMPv1 = 0
SNMP Community String	An Octet String that may contain a string used to add security to SNMP devices.
SNMP PDU	An SNMP PDU contains the body of the SNMP message. There are several types of PDUs. Three common PDUs are GetRequest, GetResponse, SetRequest.
Request ID	An Integer that identifies a particular SNMP request. This index is echoed back in the response from the SNMP agent, allowing the SNMP manager to match an incoming response to the appropriate request.
Error	An Integer set to 0x00 in the request sent by the SNMP manager. The SNMP agent places an error code in this field in the response message if an error occurred processing the request. Some error codes include:
	• 0x00 -- No error occurred
	• 0x01 -- Response message too large to transport
	• 0x02 -- The name of the requested object was not found
	• 0x03 -- A data type in the request did not match the data type in the SNMP agent
	• 0x04 -- The SNMP manager attempted to set a read-only parameter
	• 0x05 -- General Error (some error other than the ones listed above)

Error Index	If an Error occurs, the Error Index holds a pointer to the Object that caused the error, otherwise the Error Index is 0x00.
Varbind List	A Sequence of Varbinds.
Varbind	A Sequence of two fields, an Object ID and the value for/from that Object ID.
Object Identifier	An Object Identifier that points to a particular parameter in the SNMP agent.
Value	SetRequest PDU -- Value is applied to the specified OID of the SNMP agent.
	GetRequest PDU -- Value is a Null that acts as a placeholder for the return data.
	GetResponse PDU -- The returned Value from the specified OID of the SNMP agent.

GET Request Encoding

Figure below shows SNMP GetRequest packet for the microphoneMute parameter on a Rane NM 1 (OID: 1.3.6.1.4.1.2680.1.2.7.3.2.0).


SNMP Message Type = Sequence, Length = 44
SNMP Version Type = Integer Length = 1 Value = 0
SNMP Community String Type = Octet String Length = 7 Value = private
SNMP PDU Type = GetRequest, Length = 30
Request ID Type = Integer Length = 1 Value = 1
Error Type = Integer Length = 1 Value = 0
Error Index Type = Integer Length = 1 Value = 0
Varbind List Type = Sequence, Length = 19
Varbind Type = Sequence, Length = 17
Object Identifier Type = Object Identifier Length = 13 Value = 1.3.6.1.4.1.2680.1.2.7.3.2.0
Value Type = Null Length = 0
30 2C
02 01 00
04 07 70 72 69 76 61 74 65
A0 1E
02 01 01
02 01 00
02 01 00
30 13
30 11
06 0D 2B 06 01 04 01 94 78 01 02 07 03 02 00
05 00

SNMP Message Diagram

Sockets

- A socket is defined as an *endpoint for communication*.
- Concatenation of IP address and port
  - Connection-oriented: Phone number and receiver
  - Connectionless: Address and receiver
- A socket pair (local IP address, local port, foreign IP address, foreign port) uniquely identifies a communication.
- The socket **161.25.19.8:1625** refers to port **1625** on host **161.25.19.8**

Socket Programming in UDP

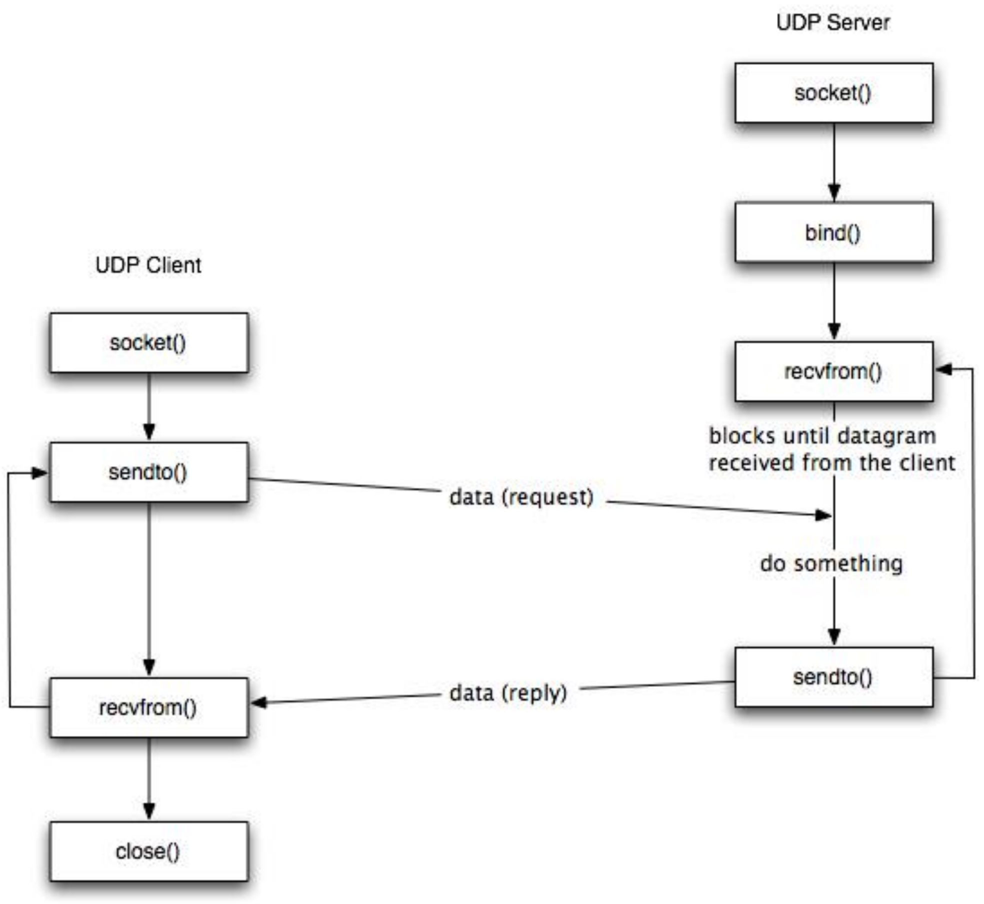
The figure below shows the interaction between a UDP client and server. The client does not establish a connection with the server. Instead, the client just sends a datagram to the server using the **sendto** function which requires the address of the destination as a parameter. Similarly, the server does not accept a connection from a client. Instead, the server just calls the **recvfrom** function, which waits until data arrives from some client. **recvfrom** returns the IP address of the client, along with the datagram, so the server can send a response to the client.

As shown in the Figure, the steps of establishing a UDP socket communication on the client side are as follows:

- Create a socket using the socket() function;
- Send and receive data by means of the recvfrom() and sendto() functions.

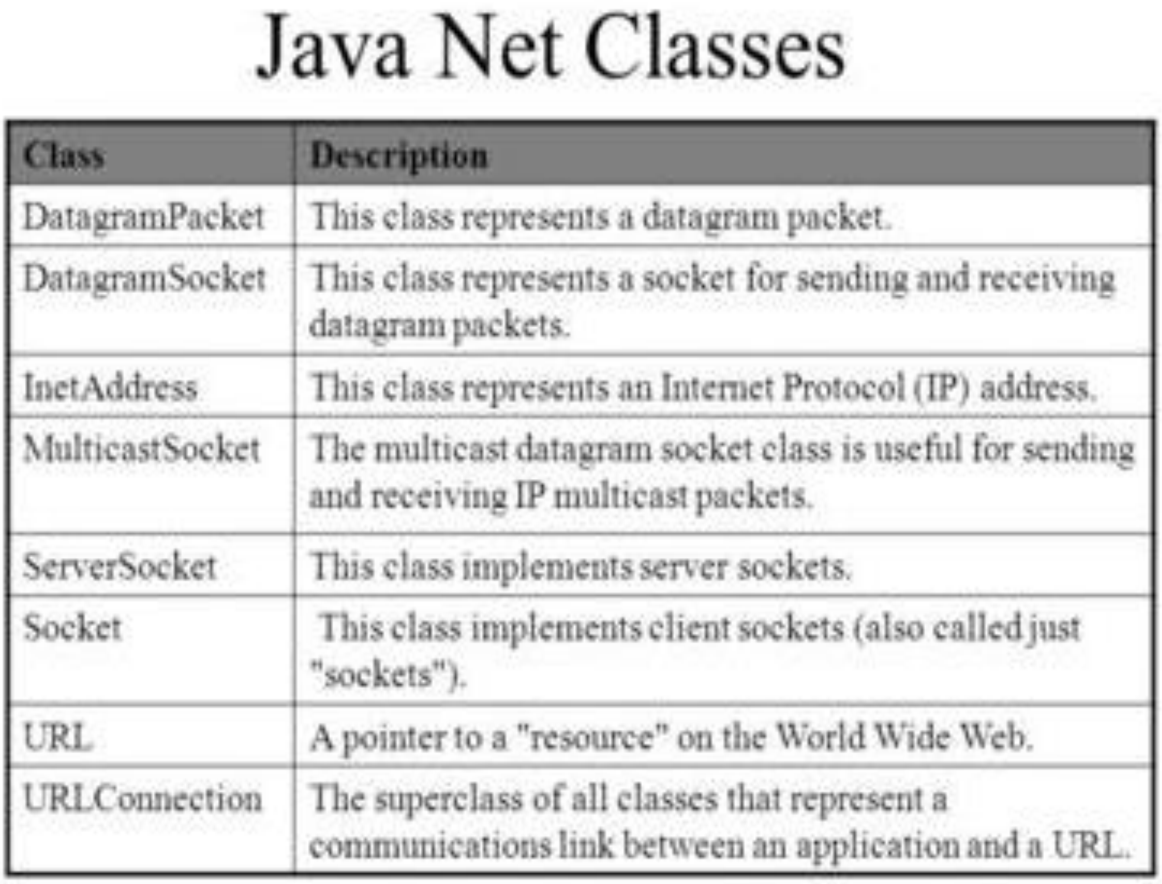
The steps of establishing a UDP socket communication on the server side are as follows:

- Create a socket with the socket() function;
- Bind the socket to an address using the bind() function;
- Send and receive data by means of recvfrom() and sendto().



Java API for UDP Datagrams

- The Java API provides datagram communication by means of two classes:
    - **DatagramPacket** - Datagram packets are used to implement a connectionless packet delivery service.
    - **DatagramSocket** - A datagram socket is the sending or receiving point for a packet delivery service.
- DatagramPacket:
    - **getData** - Returns the data buffer.
    - **getPort** - Returns the port number on the remote host.
    - **getAddress** - Returns the IP address.
- DatagramSocket:
    - **send** - Sends a datagram packet from this socket.
    - **receive** - Receives a datagram packet from this socket.
    - **setSoTimeout** - Enable/disable the specified timeout, in milliseconds.
    - **connect** - Connects the socket to a remote address for this socket.



Java Net Classes

Class	Description
DatagramPacket	This class represents a datagram packet.
DatagramSocket	This class represents a socket for sending and receiving datagram packets.
InetAddress	This class represents an Internet Protocol (IP) address.
MulticastSocket	The multicast datagram socket class is useful for sending and receiving IP multicast packets.
ServerSocket	This class implements server sockets.
Socket	This class implements client sockets (also called just "sockets").
URL	A pointer to a "resource" on the World Wide Web.
URLConnection	The superclass of all classes that represent a communications link between an application and a URL.

Implementation

Following is the implementation of SNMP Manager with GetRequest PDU in Java. The java implementation contains two modules :

i) SNMPManager.java
ii) SNMPMessage.java

<u>SNMPManager.java</u>

```
/*
 * Implementation of SNMP Manager for GetRequest PDU
 */


import java.net.*;
import java.io.*;
import java.util.Scanner;


class Dgram
{
        public static DatagramPacket toDatagram(byte buf[], InetAddress destIA, int
destPort)
        {
                return new DatagramPacket(buf, buf.length,destIA, destPort);
        }


        public static String toString(DatagramPacket p)
        {
                return new String(p.getData(), 0, p.getLength());
        }
}
```

```
public class SNMPManager
{
	private DatagramSocket s;
	private InetAddress hostAddress;
	private byte[] buf = new byte[1000];
	private byte snmp_message[];
	private DatagramPacket dp =new DatagramPacket(buf, buf.length);
	private DatagramPacket dp1 = new DatagramPacket(buf, buf.length);
	int x,y; //Temporary Variables for storing the first two bytes of OID (for iso, x = 1 and
y = 3)

	public SNMPManager(String hostname)
	{
		try
		{
			// Construct and Bind Datagram Socket to port 161
			s = new DatagramSocket(161);

			// Get IP address of host if hostname is specified
			hostAddress =InetAddress.getByName(hostname);
		}
		catch(UnknownHostException e)
		{
			System.err.println("Cannot find host");
			System.exit(1);
		}
		catch(SocketException e)
		{
			System.err.println("Can't open socket");
			e.printStackTrace();
			System.exit(1);
		}

		System.out.println("SNMP Manager is running !!\n");
	}

	// Encode the SNMP Message based on TLV format [Type,Length,Value]
	public void encode(SNMPMessage data)
	{
		/*
		 * Length of each field based on TLV format where
		 * T & L occupy the first two bytes of a field and
		 * the remaining is occupied by V which is of variable length
		 */

		// SNMP Version {0,1,2} => {v1,v2,v3}
		byte version[] = new byte[3];

		version[0] = Data_Types.INTEGER.getIdentifer();
		version[1] = 1;
		version[2] = (byte)data.getVersion();

		// SNMP Community String {public,private,...}
		byte community_string[] = new byte[data.getCommunity_String().length() +
2];

		community_string[0] = Data_Types.OCTET_STRING.getIdentifer();
		community_string[1] = (byte)data.getCommunity_String().length();

		char [] temp = data.getCommunity_String().toCharArray(); // Convert string to
a character array
		for(int i = 0; i < temp.length ;i++)	// Store character array in byte format
```

```
        community_string[2+i] = (byte)temp[i];

// Request ID {1,2,...}
byte request[] = new byte[3];

request[0] = Data_Types.INTEGER.getIdentifer();
request[1] = 1;
request[2] = (byte)data.getRequest_ID();

// Error Type {0x00, 0x01, 0x02, 0x03, 0x04, 0x05}
byte error[] = new byte[3];

error[0] = Data_Types.INTEGER.getIdentifer();
error[1] = 1;
error[2] = (byte)data.getError_Type();

// Error Index
byte error_index[] = new byte[3];

error_index[0] = Data_Types.INTEGER.getIdentifer();
error_index[1] = 1;
error_index[2] = (byte)data.getError_Index();

// Object Identifier (OID)
String[] bytes = data.getOID().split("\\.");     // Split input OID based on '.'

// Count the no. of big numbers in OID (i.e above 127)
int big_number_count = 0;
for (int i = 0; i < bytes.length; i++)
        if(Integer.decode(bytes[i]) > 127) // less than 0 condition not required
```

```
                big_number_count ++;

        byte oid[] = new byte[bytes.length + 1 + big_number_count];
        oid[0] = Data_Types.OID.getIdentifer();
        oid[1] = (byte)(bytes.length - 1 + big_number_count);

        // Parse Initial 2 bytes to encode the first number of OID according to BER
[Basic Encoding Rule]
        x = Integer.parseInt(bytes[0]);
        y = Integer.parseInt(bytes[1]);
        int first_byte = 40 * x + y;

        // Convert the first number to HEX
        String first_byte_hex = Integer.toHexString(first_byte);
        oid[2] = Integer.decode("0x"+first_byte_hex).byteValue();

        // Store the remaining numbers in OID in byte format
        for(int i = 0,j = 0; i < oid[1] && j < bytes.length - 2; i++)
            // Encode big numbers according to BER if above 127
            if(Integer.decode(bytes[2 + j]) > 127)
            {
                /*
                 * If a number in OID is 1002,  then it is encoded as 0x87 and
0x6A
                 * The first octet (first byte) is obtained by right shifting the
number by 0x07 and ORing the result with 0x80
                 * The second octet (second byte) is obtained by ANDing the
number with 0x7F
                 */
                // Counters: i -> oid array and j -> byte array (excluding the
first 2 elements)
```

```
                    oid[3 + i] = Integer.decode("0x" + 
Integer.toHexString(((Integer.decode(bytes[2 + j]) >> 0x07 )| 0x80))).byteValue(); //First 
byte

                    oid[3 + (i++) + 1] = Integer.decode("0x" + 
Integer.toHexString(Integer.decode(bytes[2 + (j++)]) & 0x7F)).byteValue();        // Second 
byte

                    // i and j is incremented within the array index

                }

                else

                    oid[3 + i] = Integer.decode(bytes[2 + (j++)]).byteValue();


            // Value

            byte value[] = new byte[2];

            value[0] = Data_Types.NULL.getIdentifer();

            value[1] = 0; // Length : 0


            // Varbind : {OID,Value}

            byte varbind[] = new byte[oid.length + value.length + 2];

            varbind[0] = Data_Types.SEQUENCE.getIdentifer();

            // Calculate length of oid and value bytearrays and convert it to HEX string 
and then to byte format

            varbind[1] = Integer.decode("0x" +Integer.toHexString(oid.length + 
value.length)).byteValue();

            System.arraycopy(oid, 0, varbind, 2, oid.length);      // Copy the byte array oid 
to byte array varbind at pos 0

            System.arraycopy(value, 0, varbind, oid.length + 2, value.length);   // Copy the 
byte array value to byte array varbind at pos "oid.length + 2"


            // Varbind List : {Varbind1, Varbind2, ...}

            byte varbindlist[] = new byte[varbind.length + 2];

            varbindlist[0] = Data_Types.SEQUENCE.getIdentifer();

            varbindlist[1] = Integer.decode("0x" + 
Integer.toHexString(varbind.length)).byteValue();

            System.arraycopy(varbind, 0, varbindlist, 2, varbind.length);
```

```
		// SNMP PDU {Request_ID, Error, Error_Index, Varbind List}

		// PDU -> {GetRequest,SetRequest,...}

		byte snmp_pdu[] = new byte[request.length + error.length + 
error_index.length + varbindlist.length + 2];

		snmp_pdu[0] = Data_Types.GETREQUEST.getIdentifer(); // For 
GET_REQUEST PDU

		snmp_pdu[1] = Integer.decode("0x" +Integer.toHexString(request.length + 
error.length + error_index.length + varbindlist.length)).byteValue();

		System.arraycopy(request, 0, snmp_pdu, 2, request.length);

		System.arraycopy(error, 0, snmp_pdu, 2 + request.length, error.length);

		System.arraycopy(error_index, 0, snmp_pdu, 2 + request.length + error.length, 
error_index.length);

		System.arraycopy(varbindlist, 0, snmp_pdu, 2 + request.length + error.length 
+ error_index.length, varbindlist.length);


		// SNMP message {SNMP Version, SNMP Community String, SNMP PDU}

		byte snmp_message[] = new byte[version.length + community_string.length + 
snmp_pdu.length + 2];

		snmp_message[0] = Data_Types.SEQUENCE.getIdentifer();

		snmp_message[1] = Integer.decode("0x" +Integer.toHexString(version.length 
+ community_string.length + snmp_pdu.length)).byteValue();

		System.arraycopy(version, 0, snmp_message, 2, version.length);

		System.arraycopy(community_string, 0, snmp_message, 2 + version.length, 
community_string.length);

		System.arraycopy(snmp_pdu, 0, snmp_message, 2 + version.length + 
community_string.length, snmp_pdu.length);


		this.snmp_message = snmp_message;

	}


	public void getRequest()

	{

		try
```

```
                    {
                              // Construct Datagram Packet with snmp message, host address and
port number
                              dp = Dgram.toDatagram(snmp_message, hostAddress,161);

                              // Send the Datagram Packet from Datagram Socket
                              s.send(dp);

                              // Prints detailed description of the message sent from the SNMP
Manager
                              SNMPMessage.print_message(snmp_message,x,y); //For OID, x=1
and y=3 ex: 1.3.6.1.2.1.1.5.0 or iso.3.6.1.2.1.1.5.0

                              // Print the message sent in string format
                              System.out.println("\nMessage sent to agent");
                              System.out.printf("Message : %s%n",Dgram.toString(dp));
                              System.out.println("\nWaiting for reply .....\n");

                              // Receive the Datagram Packet from Datagram Socket from SNMP
Agent
                              s.receive(dp1);
                              byte[] a = dp1.getData();

                              // Print the message received in string format
                              System.out.println("Message received from agent !!");
                              System.out.println("Agent IP address: " + dp1.getAddress() + ":" +
dp1.getPort());
                              System.out.printf("Message : %s%n%n",Dgram.toString(dp1));

                              // Prints detailed description of the message received from the SNMP
Agent
                              SNMPMessage.print_message(a,x,y);
```

```
		}
		catch(IOException e)
		{
			e.printStackTrace();
			System.exit(1);
		}
	}

	public static void main(String[] args)
	{
		// Read the destination hostName or hostAddress
		Scanner reader = new Scanner(System.in);

		System.out.print("Enter Destination Address/HostName: ");

		// Initialize object of SNMPManager
		SNMPManager s = new SNMPManager(reader.nextLine());

		// Initialize object of SNMPMessage
		SNMPMessage m = new SNMPMessage();
		m.getData();	// Get Input data for constructing the message from the user

		s.encode(m);	// Encode the SNMP message
		s.getRequest(); // Send GET_REQUEST to SNMP Agent and Print response

		reader.close();
	}
}
```

SNMPMessage.java

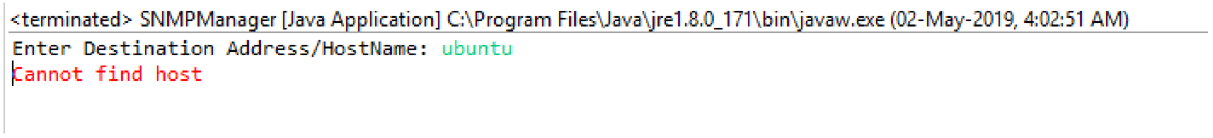
```
import java.util.Scanner;

enum Data_Types
{
	// ASN.1 Data Type and their respective identifiers
	INTEGER(0x02),OCTET_STRING(0x04),NULL(0x05),OID(0x06),SEQUENCE(0x
30),GETREQUEST(0xA0);
	private int identifier;
	private String data_type;

	Data_Types(int i)
	{
		identifier = i;
		data_type = this.name();
	}

	// To return identifier(in byte format) for specific ASN.1 Data Type
	public byte getIdentifer()
	{
		return (byte)identifier;
	}

	// To return ASN.1 Data Type for a given identifier
	public static String getData_Type(int i)
	{
		for(Data_Types x : Data_Types.values())
		{
			if(x.identifier == i)
				return x.data_type;
		}
		return "UNKNOWN";
	}
}

public class SNMPMessage
{
	private int Version,Request_ID,Error_Type,Error_Index;
	private String Community_String,OID,Value;

	// Initialize all fields of the SNMP Message
	public SNMPMessage()
	{
		setVersion(0); // SNMP Version(1) : 0
		setCommunity_String("public");	// Default community string for v1 :
public
		setRequest_ID(1);	// Request ID : 1
		setError_Type(0);	// Error Type : 0
		setError_Index(0);	// Error Index : 0
		setOID("");	// Object Identifier : NULL
		setValue(""); // Value : NULL
	}

	// Get Version, Community String and Object Identifier inputs from user
	public void getData()
	{
		System.out.println("Input SNMP Field Details");

		Scanner reader = new Scanner(System.in);

		System.out.print("Version : ");
```

```
}

        setVersion(reader.nextInt());

        reader.nextLine();

        System.out.print("Community String : ");
        setCommunity_String(reader.nextLine());

        System.out.print("OID : ");
        setOID(reader.nextLine());

        reader.close();
}

public int getVersion() {
        return Version;
}

public void setVersion(int version) {
        Version = version;
}

public String getCommunity_String() {
        return Community_String;
}

public void setCommunity_String(String community_String) {
        Community_String = community_String;
}

public int getRequest_ID() {
		return Request_ID;
	}

	public void setRequest_ID(int request_id) {
		Request_ID = request_id;
	}

	public int getError_Type() {
		return Error_Type;
	}

	public void setError_Type(int error_type) {
		Error_Type = error_type;
	}

	public int getError_Index() {
		return Error_Index;
	}

	public void setError_Index(int error_index) {
		Error_Index = error_index;
	}

	public String getOID() {
		return OID;
	}

	public void setOID(String oid) {
		OID = oid;
	}
	public String getValue() {
		return Value;
	}

	public void setValue(String value) {
		Value = value;
	}

	public static void print_message(byte[] b, int x, int y)
	{
		/*
		 * Detailed SNMP Message Description using TLV format
		 */

		String temp = "";
		int i,j,k;

		System.out.println("Message Description: \n");

		String format_title = "%1$-20s%2$-15s%3$-10s%4$-7s\n"; // Define
formatting for title of the table
		String format = "%1$-20s%2$-15s%3$-10s%4$-7s\n";	// Define
formatting for the data in the table

		System.out.format(format_title,"Field","Type","Length","Value");
		System.out.println("--------------------------------------------------------");

		// SNMP Message

	System.out.format(format,"Version",Data_Types.getData_Type(b[2]),b[3],b[4]);
```

```
                    for(i = 0; i < b[6]; i++)
                              temp += (char)b[7+i];
                    System.out.format(format,"Community
String",Data_Types.getData_Type(b[5]),b[6],temp);


                    i += 7 + 2; //    index variable for the following fields (skipping Tag and
Length for SNMP PDU)


                    // SNMP PDU
                    System.out.format(format,"Request
ID",Data_Types.getData_Type(b[i]),b[i+1],b[i+2]);

          System.out.format(format,"Error",Data_Types.getData_Type(b[i+3]),b[i+4],b[i+5]);
                    System.out.format(format,"Error
Index",Data_Types.getData_Type(b[i+6]),b[i+7],b[i+8]);


                    i += 8 + 4 + 2; //          index variable (skipping Tag and Length for Varbind
List, Varbind)


                    int bit_pos = 7, temp_x, temp_y, p = 1, m = 7;


                    for(j = 1, temp = x + "." + y; j < b[i]; j++)     //          Initial OID is "x.y"


                              //          Handles OID's with numbers upto 16383 (2^14 - 1) upto two
octets
                              if((b[i + 1 + j] & ((byte)1 << bit_pos)) != 0)  // Check if 8th bit in the
first byte of the big number is set or not
                              {
                                        temp_x = (((1 << m) - 1) & (b[i + 1 + j] >> (p - 1)));//
          Store m bits from position p in the first byte where m = 7,p = 1
                                        temp_y = b[i + 1 + j + 1];       // Store second byte
                                        temp += "." + ((temp_x * 128) + temp_y);     // Decode the
bytes into big number
```

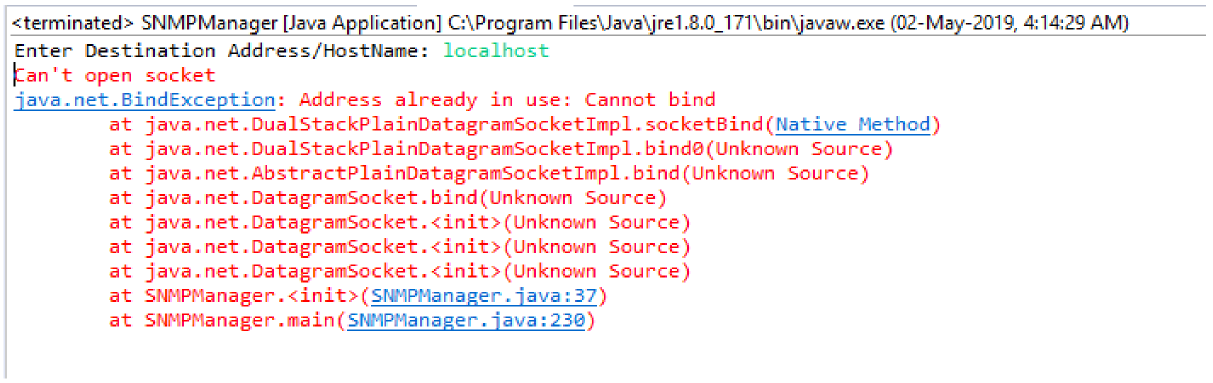
```
                                        j++;

                              }

                              else

                                        temp += "." + b[i + 1 + j];

                    System.out.format(format,"Object ID",Data_Types.getData_Type(b[i -
1]),b[i],temp);


                    for(k = 1, temp = ""; k <= b[i + 1 + j + 1]; k++)

                              if(b[i + 1 + j] == Data_Types.OCTET_STRING.getIdentifer())       //
          If value type is string, convert ASCII value to character type

                                        temp += (char)b[i + 1 + j + 1 + k];

                              else

                                        temp += b[ i + 1 + j + 1 + k];

                    System.out.format(format,"Value",Data_Types.getData_Type(b[i + 1 + j]),b[i
+ 1 + j + 1],temp);

          }

}
```

Output Screenshots



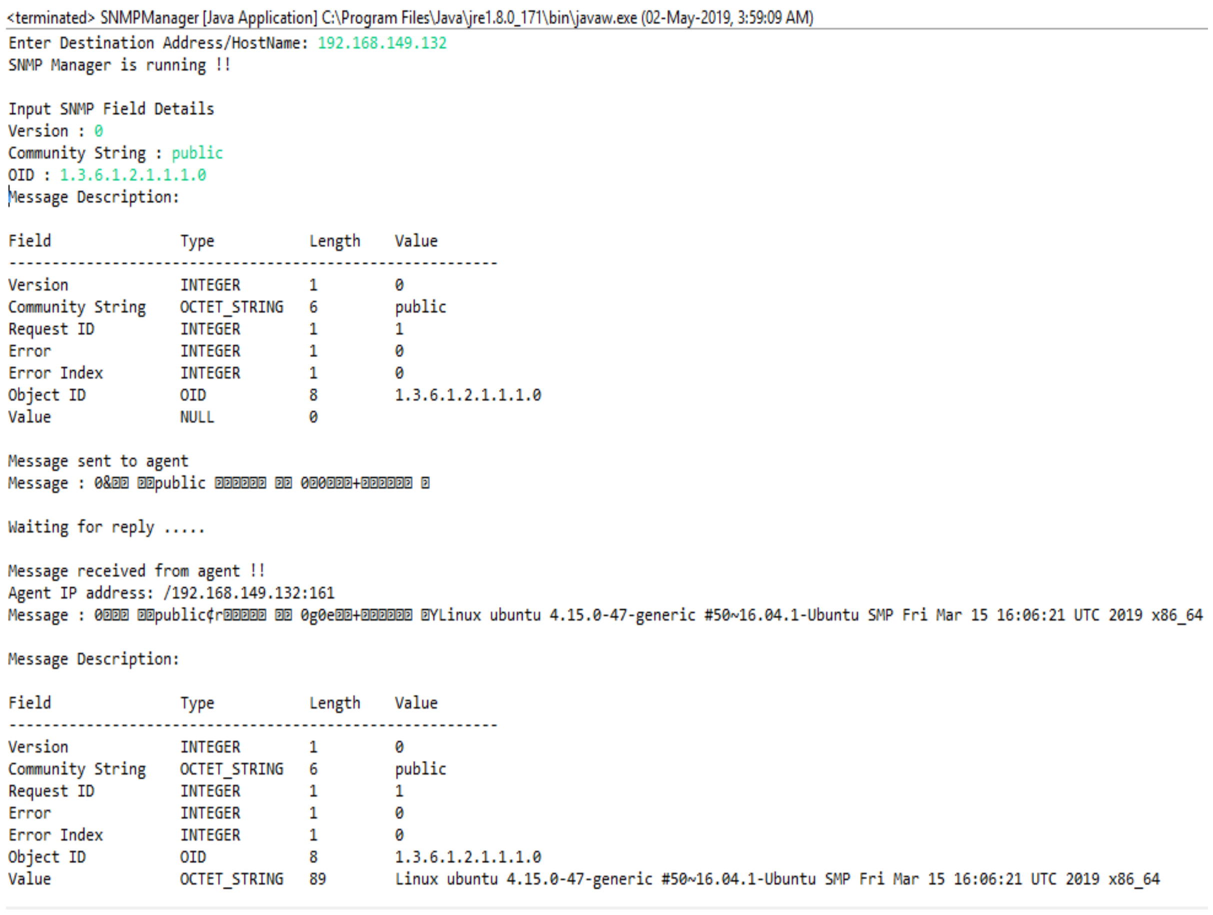
```
<terminated> SNMPManager [Java Application] C:\Program Files\Java\jre1.8.0_171\bin\javaw.exe (02-May-2019, 4:02:51 AM)
Enter Destination Address/HostName: ubuntu
Cannot find host
```

Hostname not found



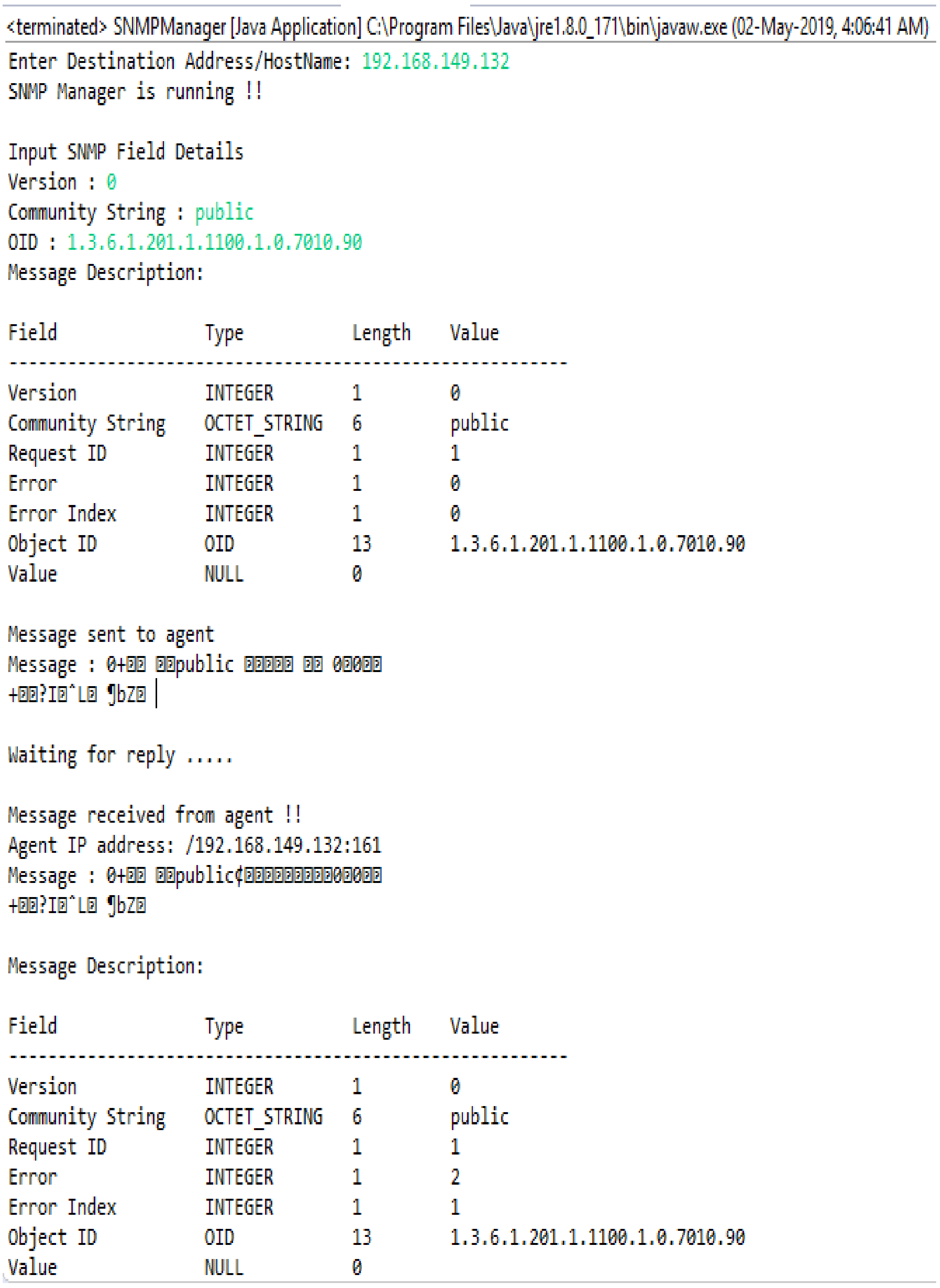
```
<terminated> SNMPManager [Java Application] C:\Program Files\Java\jre1.8.0_171\bin\javaw.exe (02-May-2019, 4:14:29 AM)
Enter Destination Address/HostName: localhost
Can't open socket
java.net.BindException: Address already in use: Cannot bind
	at java.net.DualStackPlainDatagramSocketImpl.socketBind(Native Method)
	at java.net.DualStackPlainDatagramSocketImpl.bind0(Unknown Source)
	at java.net.AbstractPlainDatagramSocketImpl.bind(Unknown Source)
	at java.net.DatagramSocket.bind(Unknown Source)
	at java.net.DatagramSocket.<init>(Unknown Source)
	at java.net.DatagramSocket.<init>(Unknown Source)
	at java.net.DatagramSocket.<init>(Unknown Source)
	at SNMPManager.<init>(SNMPManager.java:37)
	at SNMPManager.main(SNMPManager.java:230)
```

Socket in use



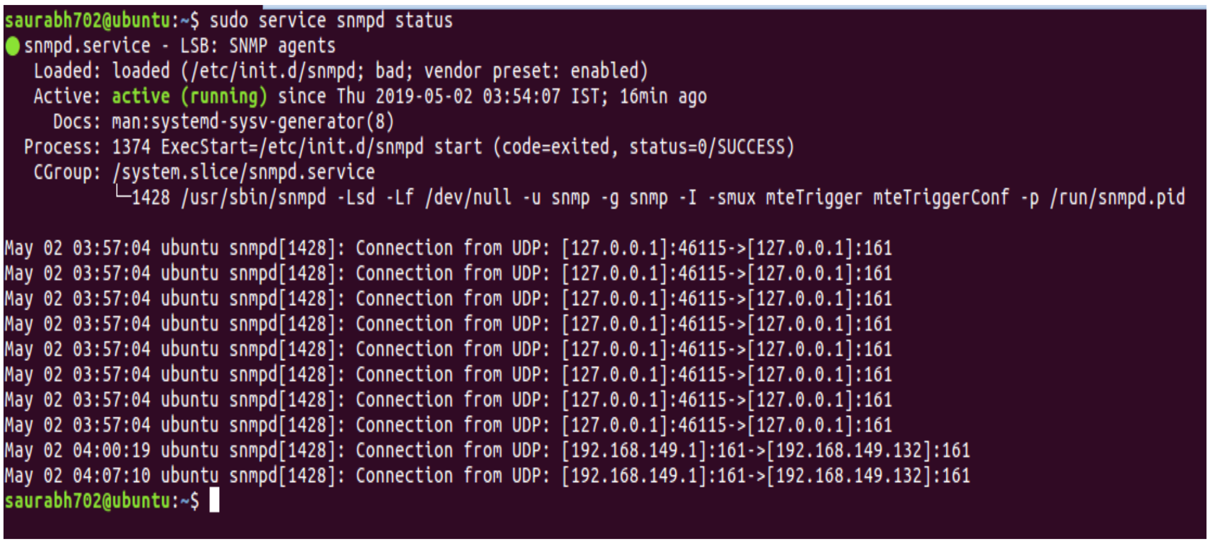
```
<terminated> SNMPManager [Java Application] C:\Program Files\Java\jre1.8.0_171\bin\javaw.exe (02-May-2019, 3:59:09 AM)
Enter Destination Address/HostName: 192.168.149.132
SNMP Manager is running !!

Input SNMP Field Details
Version : 0
Community String : public
OID : 1.3.6.1.2.1.1.1.0
Message Description:

Field               Type           Length    Value
------------------------------------------------------------
Version             INTEGER        1         0
Community String    OCTET_STRING   6         public
Request ID          INTEGER        1         1
Error               INTEGER        1         0
Error Index         INTEGER        1         0
Object ID           OID            8         1.3.6.1.2.1.1.1.0
Value               NULL           0

Message sent to agent
Message : 0&□□ □□public □□□□□□ □□ 0□0□□□+□□□□□□ □

Waiting for reply .....

Message received from agent !!
Agent IP address: /192.168.149.132:161
Message : 0□□□ □□public¢r□□□□□ □□ 0g0e□□+□□□□□□ □YLinux ubuntu 4.15.0-47-generic #50~16.04.1-Ubuntu SMP Fri Mar 15 16:06:21 UTC 2019 x86_64

Message Description:

Field               Type           Length    Value
------------------------------------------------------------
Version             INTEGER        1         0
Community String    OCTET_STRING   6         public
Request ID          INTEGER        1         1
Error               INTEGER        1         0
Error Index         INTEGER        1         0
Object ID           OID            8         1.3.6.1.2.1.1.1.0
Value               OCTET_STRING   89        Linux ubuntu 4.15.0-47-generic #50~16.04.1-Ubuntu SMP Fri Mar 15 16:06:21 UTC 2019 x86_64
```

Response from Agent



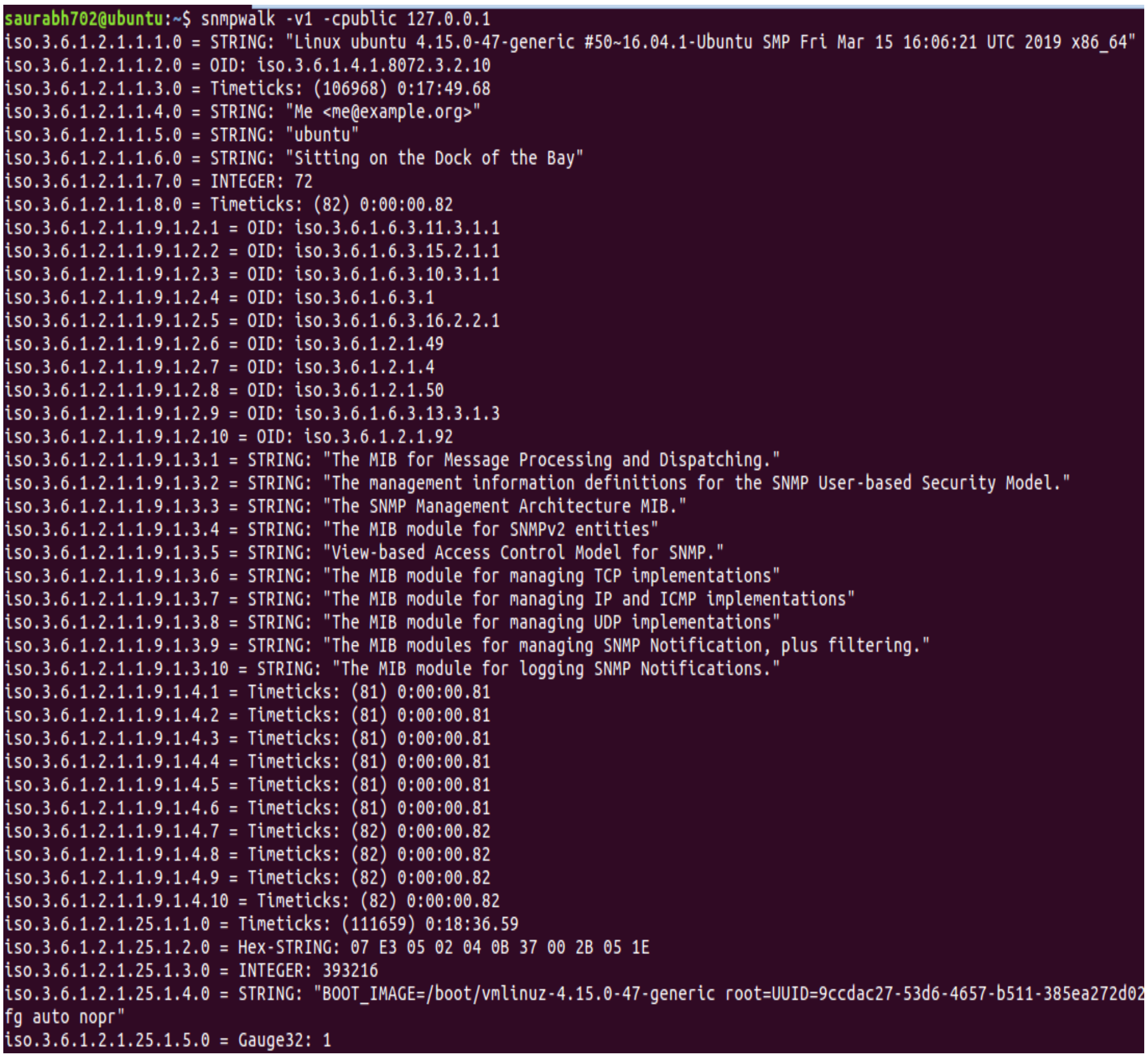
```
<terminated> SNMPManager [Java Application] C:\Program Files\Java\jre1.8.0_171\bin\javaw.exe (02-May-2019, 4:06:41 AM)
Enter Destination Address/HostName: 192.168.149.132
SNMP Manager is running !!

Input SNMP Field Details
Version : 0
Community String : public
OID : 1.3.6.1.201.1.1100.1.0.7010.90
Message Description:

Field               Type          Length    Value
---------------------------------------------------------
Version             INTEGER       1         0
Community String    OCTET_STRING  6         public
Request ID          INTEGER       1         1
Error               INTEGER       1         0
Error Index         INTEGER       1         0
Object ID           OID           13        1.3.6.1.201.1.1100.1.0.7010.90
Value               NULL          0

Message sent to agent
Message : 0+⍰⍰ ⍰⍰public ⍰⍰⍰⍰⍰ ⍰⍰ 0⍰0⍰⍰
+⍰⍰?I⍰ˆL⍰ ¶bZ⍰ |

Waiting for reply .....

Message received from agent !!
Agent IP address: /192.168.149.132:161
Message : 0+⍰⍰ ⍰⍰public¢⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰⍰0⍰0⍰⍰
+⍰⍰?I⍰ˆL⍰ ¶bZ⍰

Message Description:

Field               Type          Length    Value
---------------------------------------------------------
Version             INTEGER       1         0
Community String    OCTET_STRING  6         public
Request ID          INTEGER       1         1
Error               INTEGER       1         2
Error Index         INTEGER       1         1
Object ID           OID           13        1.3.6.1.201.1.1100.1.0.7010.90
Value               NULL          0
```

Error in OID



```
saurabh702@ubuntu:~$ sudo service snmpd status
● snmpd.service - LSB: SNMP agents
   Loaded: loaded (/etc/init.d/snmpd; bad; vendor preset: enabled)
   Active: active (running) since Thu 2019-05-02 03:54:07 IST; 16min ago
     Docs: man:systemd-sysv-generator(8)
  Process: 1374 ExecStart=/etc/init.d/snmpd start (code=exited, status=0/SUCCESS)
   CGroup: /system.slice/snmpd.service
           └─1428 /usr/sbin/snmpd -Lsd -Lf /dev/null -u snmp -g snmp -I -smux mteTrigger mteTriggerConf -p /run/snmpd.pid

May 02 03:57:04 ubuntu snmpd[1428]: Connection from UDP: [127.0.0.1]:46115->[127.0.0.1]:161
May 02 03:57:04 ubuntu snmpd[1428]: Connection from UDP: [127.0.0.1]:46115->[127.0.0.1]:161
May 02 03:57:04 ubuntu snmpd[1428]: Connection from UDP: [127.0.0.1]:46115->[127.0.0.1]:161
May 02 03:57:04 ubuntu snmpd[1428]: Connection from UDP: [127.0.0.1]:46115->[127.0.0.1]:161
May 02 03:57:04 ubuntu snmpd[1428]: Connection from UDP: [127.0.0.1]:46115->[127.0.0.1]:161
May 02 03:57:04 ubuntu snmpd[1428]: Connection from UDP: [127.0.0.1]:46115->[127.0.0.1]:161
May 02 03:57:04 ubuntu snmpd[1428]: Connection from UDP: [127.0.0.1]:46115->[127.0.0.1]:161
May 02 03:57:04 ubuntu snmpd[1428]: Connection from UDP: [127.0.0.1]:46115->[127.0.0.1]:161
May 02 04:00:19 ubuntu snmpd[1428]: Connection from UDP: [192.168.149.1]:161->[192.168.149.132]:161
May 02 04:07:10 ubuntu snmpd[1428]: Connection from UDP: [192.168.149.1]:161->[192.168.149.132]:161
saurabh702@ubuntu:~$
```

SNMP Agent status



```
saurabh702@ubuntu:~$ snmpwalk -v1 -cpublic 127.0.0.1
iso.3.6.1.2.1.1.1.0 = STRING: "Linux ubuntu 4.15.0-47-generic #50~16.04.1-Ubuntu SMP Fri Mar 15 16:06:21 UTC 2019 x86_64"
iso.3.6.1.2.1.1.2.0 = OID: iso.3.6.1.4.1.8072.3.2.10
iso.3.6.1.2.1.1.3.0 = Timeticks: (106968) 0:17:49.68
iso.3.6.1.2.1.1.4.0 = STRING: "Me <me@example.org>"
iso.3.6.1.2.1.1.5.0 = STRING: "ubuntu"
iso.3.6.1.2.1.1.6.0 = STRING: "Sitting on the Dock of the Bay"
iso.3.6.1.2.1.1.7.0 = INTEGER: 72
iso.3.6.1.2.1.1.8.0 = Timeticks: (82) 0:00:00.82
iso.3.6.1.2.1.1.9.1.2.1 = OID: iso.3.6.1.6.3.11.3.1.1
iso.3.6.1.2.1.1.9.1.2.2 = OID: iso.3.6.1.6.3.15.2.1.1
iso.3.6.1.2.1.1.9.1.2.3 = OID: iso.3.6.1.6.3.10.3.1.1
iso.3.6.1.2.1.1.9.1.2.4 = OID: iso.3.6.1.6.3.1
iso.3.6.1.2.1.1.9.1.2.5 = OID: iso.3.6.1.6.3.16.2.2.1
iso.3.6.1.2.1.1.9.1.2.6 = OID: iso.3.6.1.2.1.49
iso.3.6.1.2.1.1.9.1.2.7 = OID: iso.3.6.1.2.1.4
iso.3.6.1.2.1.1.9.1.2.8 = OID: iso.3.6.1.2.1.50
iso.3.6.1.2.1.1.9.1.2.9 = OID: iso.3.6.1.6.3.13.3.1.3
iso.3.6.1.2.1.1.9.1.2.10 = OID: iso.3.6.1.2.1.92
iso.3.6.1.2.1.1.9.1.3.1 = STRING: "The MIB for Message Processing and Dispatching."
iso.3.6.1.2.1.1.9.1.3.2 = STRING: "The management information definitions for the SNMP User-based Security Model."
iso.3.6.1.2.1.1.9.1.3.3 = STRING: "The SNMP Management Architecture MIB."
iso.3.6.1.2.1.1.9.1.3.4 = STRING: "The MIB module for SNMPv2 entities"
iso.3.6.1.2.1.1.9.1.3.5 = STRING: "View-based Access Control Model for SNMP."
iso.3.6.1.2.1.1.9.1.3.6 = STRING: "The MIB module for managing TCP implementations"
iso.3.6.1.2.1.1.9.1.3.7 = STRING: "The MIB module for managing IP and ICMP implementations"
iso.3.6.1.2.1.1.9.1.3.8 = STRING: "The MIB module for managing UDP implementations"
iso.3.6.1.2.1.1.9.1.3.9 = STRING: "The MIB modules for managing SNMP Notification, plus filtering."
iso.3.6.1.2.1.1.9.1.3.10 = STRING: "The MIB module for logging SNMP Notifications."
iso.3.6.1.2.1.1.9.1.4.1 = Timeticks: (81) 0:00:00.81
iso.3.6.1.2.1.1.9.1.4.2 = Timeticks: (81) 0:00:00.81
iso.3.6.1.2.1.1.9.1.4.3 = Timeticks: (81) 0:00:00.81
iso.3.6.1.2.1.1.9.1.4.4 = Timeticks: (81) 0:00:00.81
iso.3.6.1.2.1.1.9.1.4.5 = Timeticks: (81) 0:00:00.81
iso.3.6.1.2.1.1.9.1.4.6 = Timeticks: (81) 0:00:00.81
iso.3.6.1.2.1.1.9.1.4.7 = Timeticks: (82) 0:00:00.82
iso.3.6.1.2.1.1.9.1.4.8 = Timeticks: (82) 0:00:00.82
iso.3.6.1.2.1.1.9.1.4.9 = Timeticks: (82) 0:00:00.82
iso.3.6.1.2.1.1.9.1.4.10 = Timeticks: (82) 0:00:00.82
iso.3.6.1.2.1.25.1.1.0 = Timeticks: (111659) 0:18:36.59
iso.3.6.1.2.1.25.1.2.0 = Hex-STRING: 07 E3 05 02 04 0B 37 00 2B 05 1E
iso.3.6.1.2.1.25.1.3.0 = INTEGER: 393216
iso.3.6.1.2.1.25.1.4.0 = STRING: "BOOT_IMAGE=/boot/vmlinuz-4.15.0-47-generic root=UUID=9ccdac27-53d6-4657-b511-385ea272d02
fg auto nopr"
iso.3.6.1.2.1.25.1.5.0 = Gauge32: 1
```

MIB List

References

- Google.com
- techopedia.com
- “SNMP: Simple? Network Management Protocol”, Rane.com
- “Network Management and SNMP”, intronetworks.cs.luc.edu
- Wikipedia.com
- searchnetworking.techtarget.com
- “NETCONF/YANG”, fir3net.com
- “Socket Programming”, cs.dartmouth.edu